Université de Nice - Sophia Antipolis

Polytech Nice Sophia

Sciences Informatiques 5ème année

SOA

Rapport final de projet : Blue Galactic X

Equipe J

David BISEGNA, Jason HAENLIN, Yassine JRAD, Adrien VASSEUR, Betsara MARCELLIN

Enseignants : P.Collet, L.Gaillard, M.Chevalier

Table des matières

I. Notre compréhension du sujet

A. Acteurs

Nous avons en tout 7 acteurs qui vont intervenir. Ils sont présentés ci-dessous :

- Richard : Le commandant de mission qui va se préoccuper du bon déroulement de la mission, en exploitant les informations qui lui sont transmises par les autres départements.
- Elon : Le chef du département Rocket qui va être responsable de l'état de la fusée jusqu'au départ de cette dernière puis va pouvoir effectuer une commande sur la fusée (séparer la fusée en deux)
- Tory : L'officier du département Weather s'occupe de rendre compte des conditions météorologiques au commandant de mission.
- Jeff : L'officier du département Telemetry récupère et enregistre les informations de la fusée, de ses boosters et du payload pendant toute la durée de la mission.
- Gwynne : Le chef du département Payload consulte les informations de la charge (payload) transportées pour savoir si ce dernier a atteint et rester sur la bonne orbite.
- Peter : Le PDG de Blue Origin X veut savoir si le booster est revenu à la base ou non.
- Marie : Le webcaster consulte les informations sur la mission et la retransmet sur le web
- Un "Scheduler" qui s'occupe d'ajouter une notion de temps, ce qui nous permet de pouvoir réellement visualiser les données télémétriques qui évoluent dans le temps et ainsi avoir des comportements automatiques en fonction de ces métriques (par exemple la séparation entre la fusée et les boosters lorsqu'on atteint un certain niveau d'essence). Sans ça, nous serions obligés de changer l'état de tous nos composants instantanément entre chaque action.

B. Choix réalisés par rapport aux nouvelles US

En fonction des nouvelles US et de notre ancienne architecture, nous avons décidé d'effectuer plusieurs choix qui sont listés ci-dessous.

La phase de séparation de fusée se fait lorsque la fusée a atteint une certaine distance.

Lorsque les boosters sont en phase d'atterrissage, leur vitesse est réduite (amortissement) afin que les boosters ne s'écrasent pas au moment d'atterrir.

On a rajouté une phase de déploiement pour payload qui est géré par le web service "Orbital-payload"

Lors du début de la mission, les départements rockets et weather ne produisent plus un rapport détaillé et envoie juste un message “GO” ou “NO-GO” . Ainsi Richard donne l’autorisation à Elon pour lancer la rocket.

Les anomalies sont gérées automatiquement sans l’intervention d’un persona, un nouveau service “anomaly” va être en constante écoute sur les télémétries provenant de tous les départements concernés (Booster, Rocket, Payload), il détecte ainsi un problème et peut alerter d’une anomalie. Si une anomalie grave est détectée, un nouveau web service “Destroyer” qui est en écoute sur “anomaly” se charge de détruire la fusée.

Afin que Richard puisse consulter les informations sur la mission, nous avons ajouté un service “MissionLogReader” qui permet de lire les dernières informations provenant de la mission. “MissionLogWriter” se charge d’écrire les logs avec les informations intéressantes pour Richard en évitant d’ajouter des informations qui ne lui sont pas destinées en écoutant les évènements générés par les autres services.

US Manquantes dans notre architecture :

- Le persona “Marie” n’est pas pris en compte (US 15)
- On ne peut pas gérer plusieurs lancement de fusées en parallèles (US 16)

II. Architecture actuelle

Nous avons un système composé de plusieurs web-services écrit en spring-boot, une gestion de la persistance avec des bases de données MongoDB. Un client nous sert à réaliser les tests fonctionnels inter-webservice et les tests d’intégrations, chaque web service ont des tests unitaires et fonctionnels liés à leurs responsabilités. Tout cela est orchestré sur un CI Jenkins.

A. Web services associés

Weather Web Service

Ce service permet de vérifier la météo actuelle.

Booster Web Service

Ce service permet de créer et gérer les boosters liés à la fusée (gère la vitesse lors du passage à l’altitude MaxQ et l'atterrissage d'un booster) et s’occupe des communications entre les boosters (physique) et le serveur.

Rocket Web Service

Ce service permet de gérer les fusées (gère le lancement et effectue la séparation d'une fusée) et s'occupe des communications entre la fusée (physique) et le serveur.

Telemetry writer Web Service

Ce service permet d'écrire dans une BDD les télémétries venant de booster, rocket et payload.

Telemetry reader Web Service

Ce service permet de lire dans une BDD les télémétries écrites par "Telemetry writer".

Payload Web Service

Ce service permet de définir ce qu'est un payload. Chaque payload est associé à une fusée, un payload et un objectif (coordonnée) et gère les communications des différents payload (physique) et le serveur.

Anomaly Web Service

Ce service permet d'alerter d'une anomalie si et seulement si une alerte est détectée lors de la lecture des télémétries.

Module Destroyer Web Service

Ce service donne l'ordre de destruction si une anomalie grave est signalée.

MissionLogWriter Web Service

Ce service permet d'écrire des logs sur l'état d'avancement de la mission.

MissionLogReader Web Service

Ce service permet de lire les logs sur l'état d'avancement de la mission.

MissionPreparation Web Service

Ce service défini ce qu'est une mission, chaque mission est associée à une rocket, des boosters, un payload et un objectif (coordonnée). Permet également de lancer l'ordre pour que la mission démarre (GO/NO-GO).

MissionControl Web Service

Ce service permet de gérer l'état d'une mission du lancement (démarrage de la fusée) jusqu'à la fin de la mission (lorsque le satellite est délivré dans l'espace).

Orbital payload Web Service

Ce service permet de gérer le payload lorsque celui-ci se sépare de la fusée, (indique si le payload est arrivé à destination).

B. Les différents topics kafka

team.j.anomaly : Permet d'envoyer les événements liés aux anomalies
teamj.rocket.status : Permet d'envoyer les événements liés aux statuts des rockets
teamj.payload.status : Permet d'envoyer les événements liés aux statuts des payloads
teamj.department-status : Permet d'envoyer les événements liés GO/NO-GO
teamj.booster.status : Permet d'envoyer les événements liés aux statuts des boosters
teamj.module-destruction : Permet d'envoyer les événements liés à une destruction de fusée et de propulseur
teamj.mission-preparation : Permet d'envoyer les événements liés à une nouvelle mission
teamj.telemetry-payload : Permet d'envoyer les événements liés aux télémétries de payload
teamj.telemetry-booster : Permet d'envoyer les événements liés aux télémétries de booster
teamj.telemetry-rocket : Permet d'envoyer les événements liés aux télémétries de booster
teamj.booster-landingstep : Permet d'envoyer les événements liés à l'atterrissage d'un booster

C. Plusieurs vues de l'architecture

Lancement de la mission (voir ce lien pour zoomer)

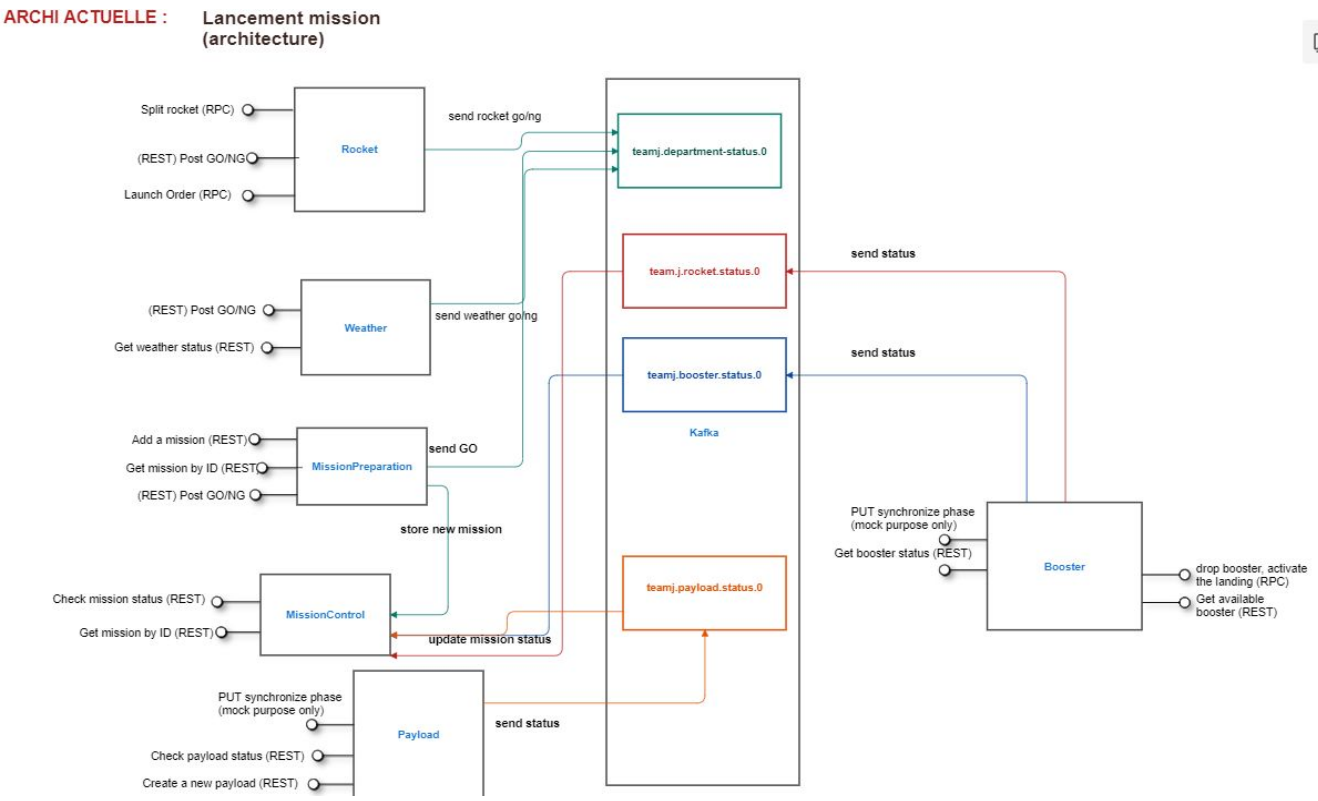

Figure 1 : diagramme de service de lancement de mission

Gestion des anomalies (voir ce lien pour zoomer)

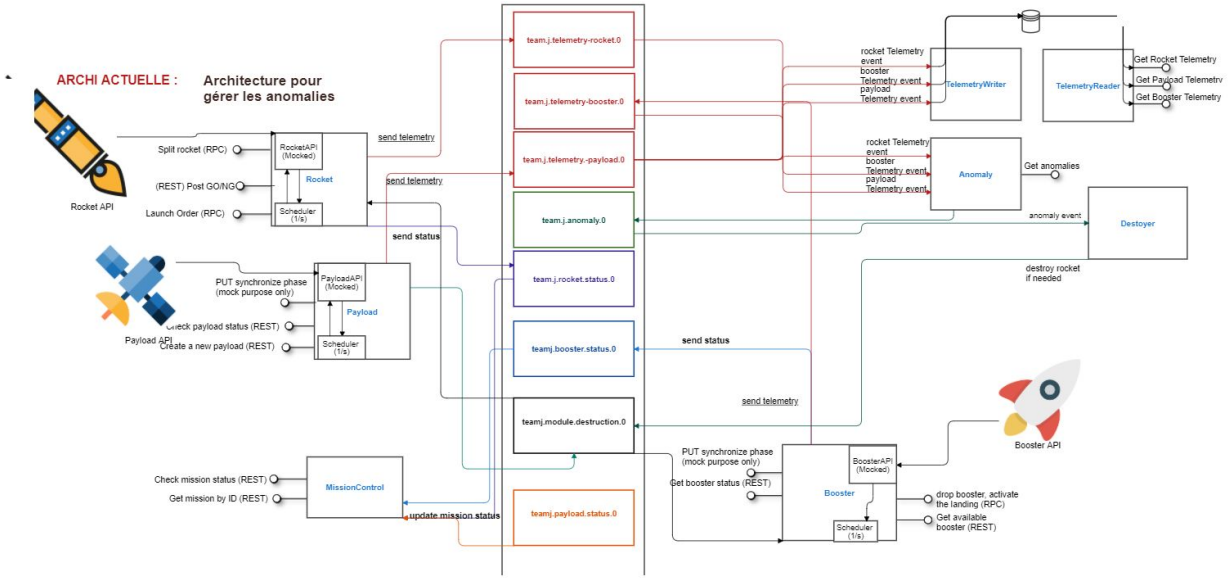

Figure 2 : diagramme de service de gestion des anomalies

Gestion de maxQ, la phase d'atterrissage du premier booster et de la phase de déploiement (voir ce lien pour zoomer)

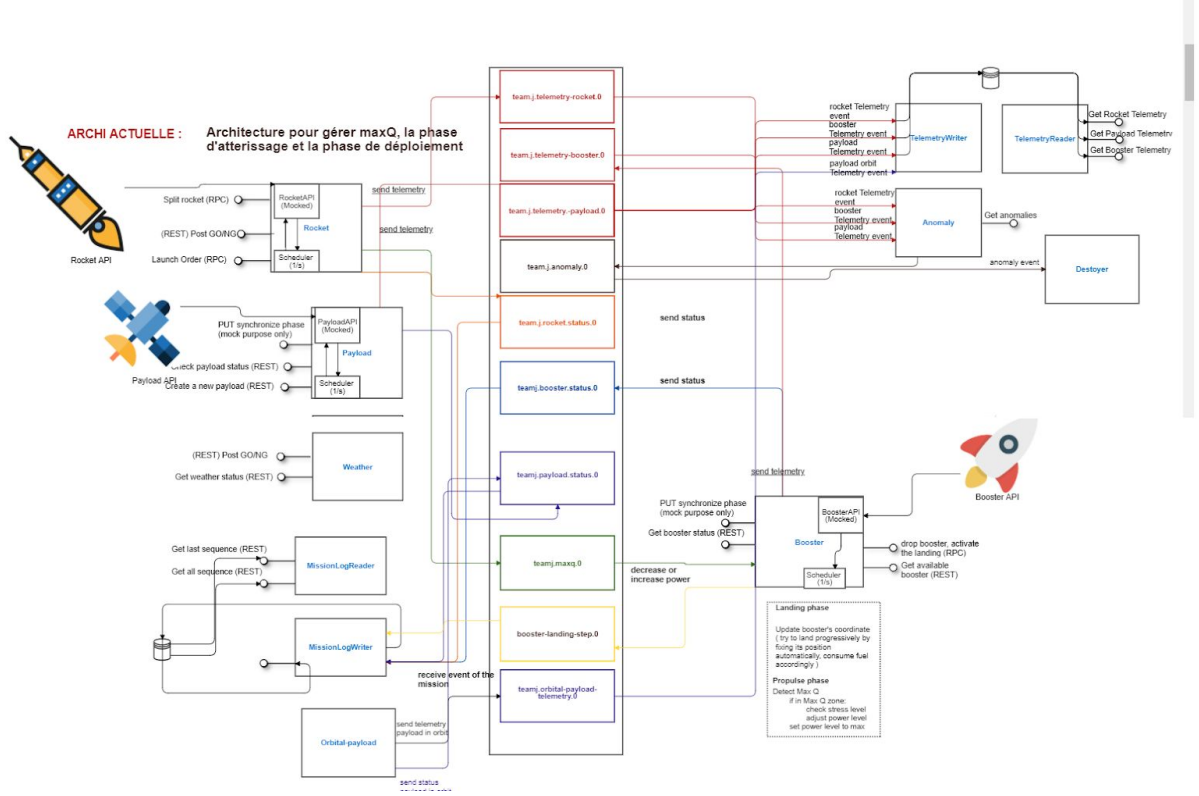

Figure 3 : diagramme de service de gestion de MaxQ, l'atterrissage du propulseur et le déploiement de payload

D. Limites et Faiblesses de l'architecture

WebServices

Rocket Web Service

Ce service gère actuellement beaucoup trop de métiers, ce qui le rend proche d'un "macro-service". Le lancement d'une fusée, la séparation d'une fusée et gérer la communication avec la fusée sont trois métiers bien distincts.

Booster Web Service

Même constat que pour rocket web service, un même service gère l'atterrissage, la création de boosters et contrôle la vitesse des boosters lors de la phase de maxQ, mais cela devrait être trois services distincts.

MissionPreparation Web Service

Actuellement ce service permet d'assembler toutes les informations pour démarrer une mission (assemble le payload, les boosters et la rocket) puis s'assure que la mission peut démarrer (vérifie le GO/NO-GO des départements rocket et weather). Mais l'assemblage et la vérification qu'une mission peut démarrer sont deux métiers différents, une seule équipe a deux responsabilités ici.

Globalement

- Le découpage pas encore assez "micro"
- Communications inter-web service très couplés (Grpc), il faut constamment mettre à jour les fichiers protobuf lors d'un changement

III. Difficultés rencontrées

Nous avons rencontré plusieurs difficultés lors de la deuxième phase du projet (après le MVP). Ce sont principalement les difficultés techniques et une mauvaise stratégie de tests qui nous ont ralenti. Le découpage tardif des services nous a également ralenti lors de la fin du projet.

Nous avons tout d'abord rencontré des difficultés pour intégrer kafka. Puis nous avons rencontré des difficultés pour tester l'intégration de micro-service. Nous n'avions pas mis en place une stratégie assez efficace pour tester l'intégration des micro-services, ce qui nous a ralenti pour faire la démo, car on se retrouvait à gérer des bugs lors de la démo qui n'était pas visible auparavant.

IV. Scénarios et diagrammes de séquences

Scénario 1 : Préparer une mission

Avant de préparer une mission, les informations de la fusée, des boosters et du payload sont créées et stockées. Suite à ça Richard peut récupérer via L'API de chaque webservice les informations dont il a besoin pour créer une mission avec toutes les informations nécessaires.

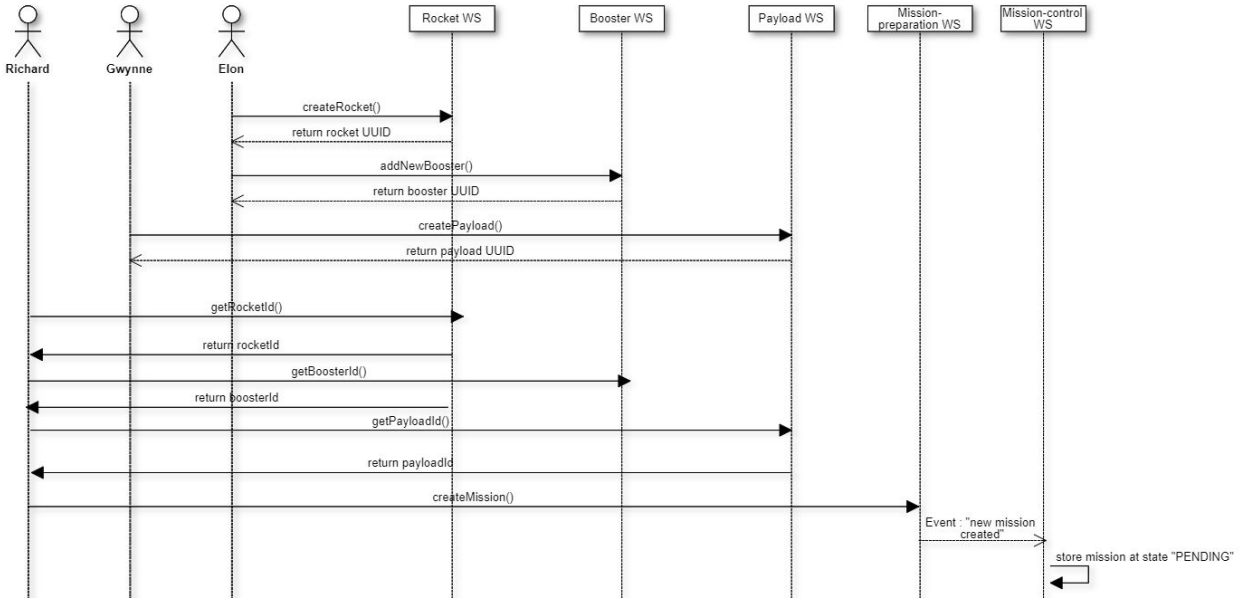

Figure 4 : diagramme de séquence pour préparer une mission

Scénario 2 : GO/NO GO poll avant le départ [User story : 1, 2, 3 et 4]

Afin de pouvoir démarrer la fusée, Richard au centre du déroulement de la mission doit vérifier que tout est prêt pour un lancement de la fusée. En amont, Tory consulte régulièrement la météo (précipitation, vent, …), et après avoir consulté ces données, il peut signaler le département Mission. Elon quant à lui, va consulter régulièrement l'état de la fusée (vibration, température, …) et signaler le département Mission. Richard de son côté fait des vérifications régulières, pour voir si le département Weather et département Rocket ont tous les deux signalés "GO" pour leurs départements, ensuite, Richard peut envoyer à son tour un signalement "GO" pour son département. Elon peut maintenant enclencher le lancement de la fusée.

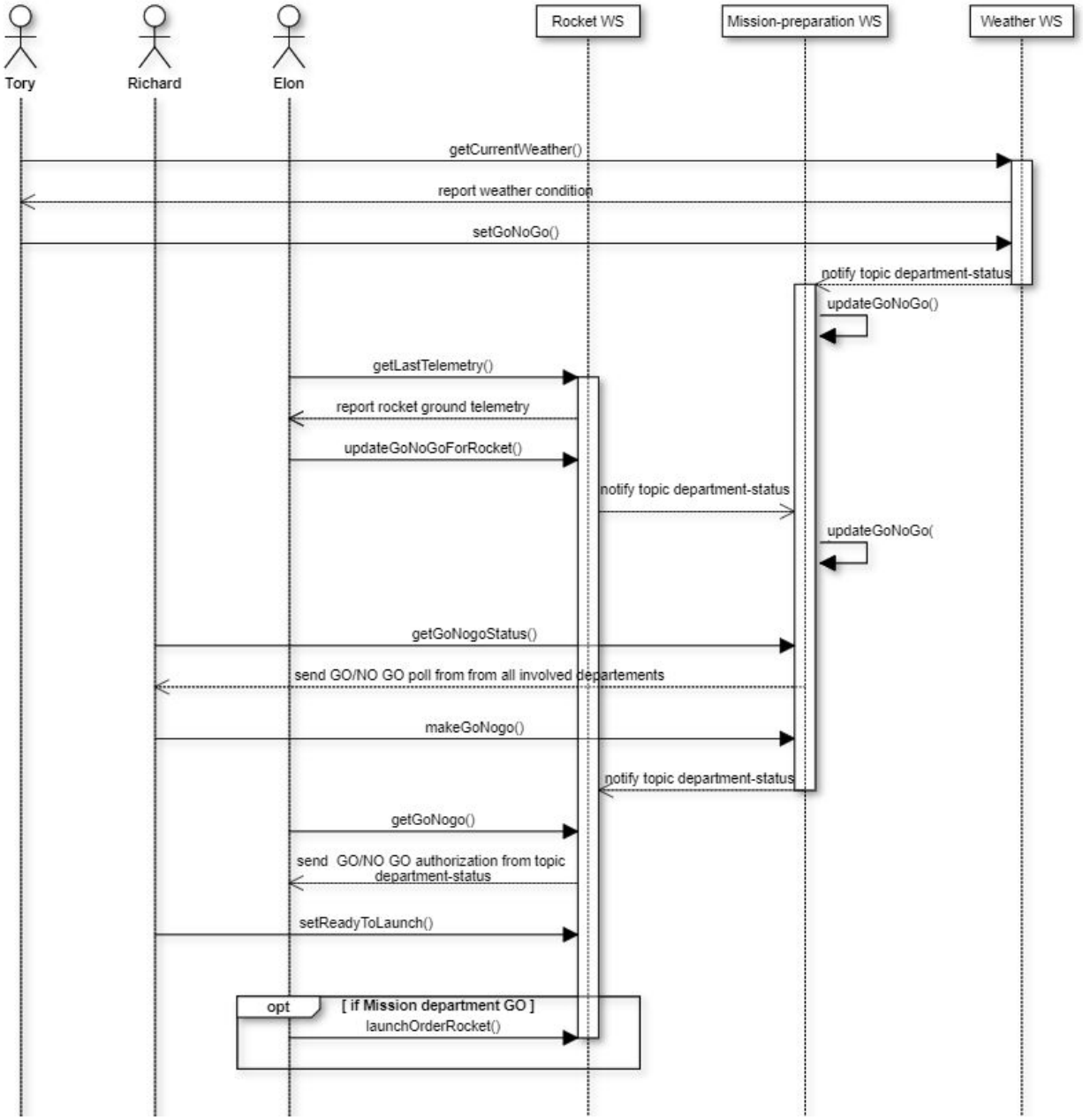

Figure 5 : diagramme de séquence du poll GO/NO GO

Scénario 3 : Lancement de la fusée [User story : 16]

Afin de pouvoir lancer la fusée, Elon lance un ordre pour commencer le démarrage de la fusée, suite à ça une requête en interne est lancée pour commencer à démarrer les boosters et un évènement est lancé pour mettre à jour le système. Lorsque les boosters ont démarré, Elon lance un ordre de lancement pour que la fusée décolle, un ordre aux boosters est envoyé en interne pour qu'ils se propulsent et un événement est envoyé pour mettre à jour tout le système .

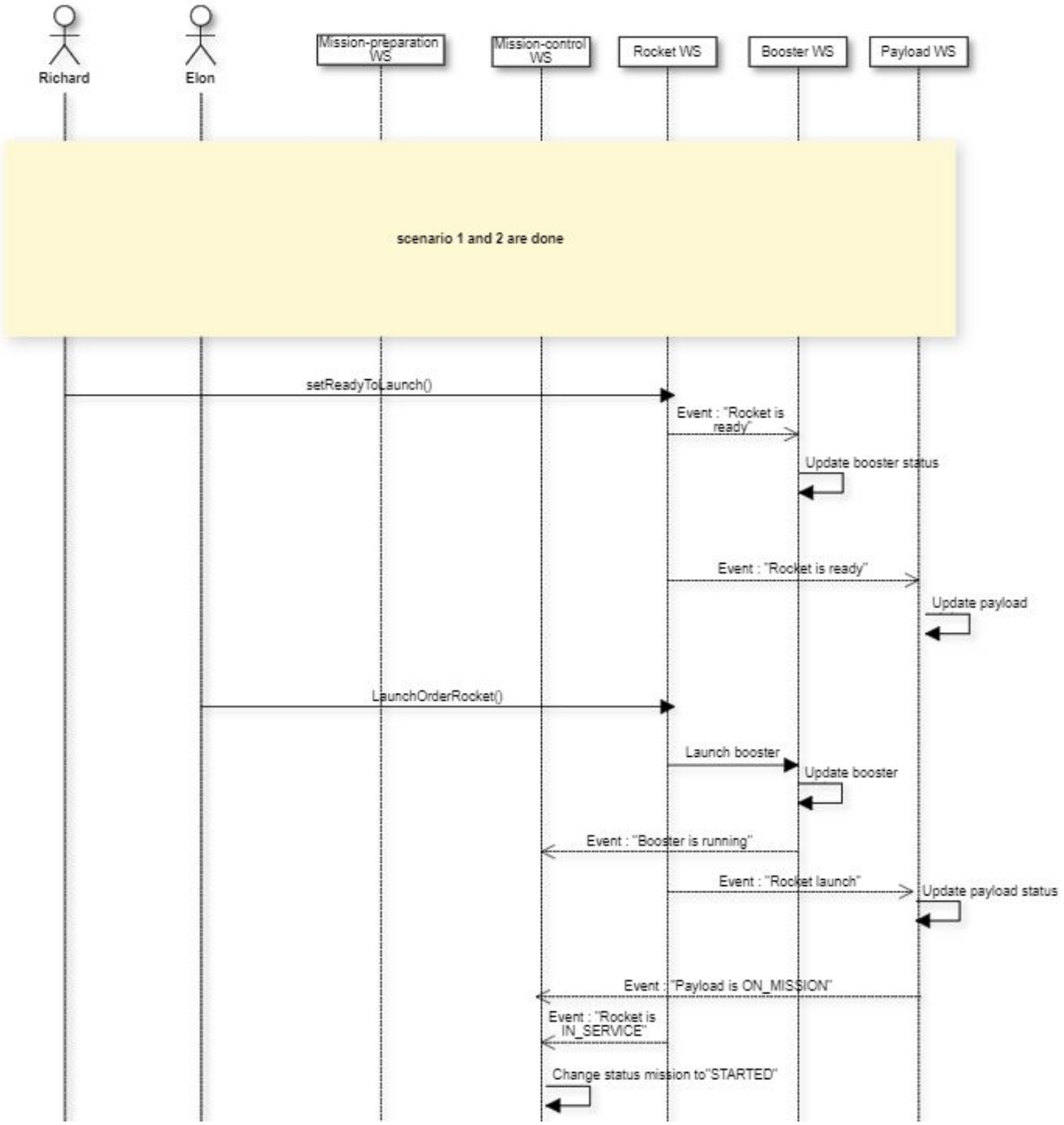

Figure 6 : diagramme de séquence de lancement de la fusée

Scénario 4 : Faire revenir le propulseur à la base [User story : 6, 9, 10, 13]

NB: maintenant le stage de la rocket se fait automatiquement (cf. user story 6), Elon n'intervient plus dans cette étape. Cependant si Elon le souhaite il peut envoyer la requête RPC vers Booster WS comme le fait le "simulateur de Rocket".

Une fois que le propulseur (booster) s'est détaché de la fusée, elle rentre dans une phase "LANDING". Au cours de cette phase, les télémétries du propulseur sont régulièrement envoyées au département Telemetry, qui sont consultables par Jeff. En même temps, au fil de la descente du propulseur, ce dernier passe par plusieurs étapes (dit state) en accordance avec

l'altitude où le propulseur se situe par rapport à la station d'atterrissage : FLIPPING, ENTRY_BURN, LANDING_BURN, LEGS_DEPLOYED et LANDED.

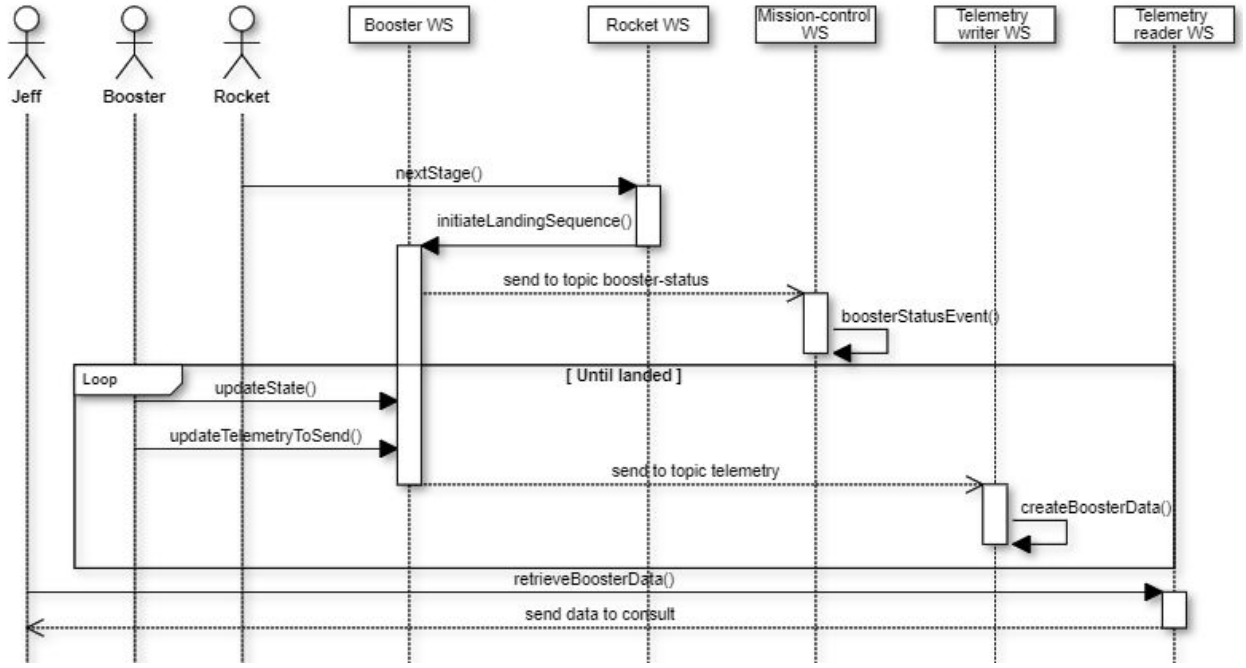

Figure 7 : diagramme de séquence d'atterrissage du propulseur

Scénario 5 : Gérer les anomalies [US 18, 17, 8]

Au cours du vol de la fusée, les "modules" qui y sont attachés (propulseur, fusée, payload) peuvent rencontrer des problèmes. Les télémétries qui sont envoyées par ces modules sont analysées en permanence par notre système. Lorsque notre système détecte l'anomalie, il va pouvoir décider des actions à mener en conséquence, si l'anomalie met en danger la vie des personnes sur terre, le module est détruit automatiquement. Malgré tout, Richard peut tout de même décider de détruire la fusée s'il estime que l'anomalie identifiée est grave, de part sa connaissance des anomalies.

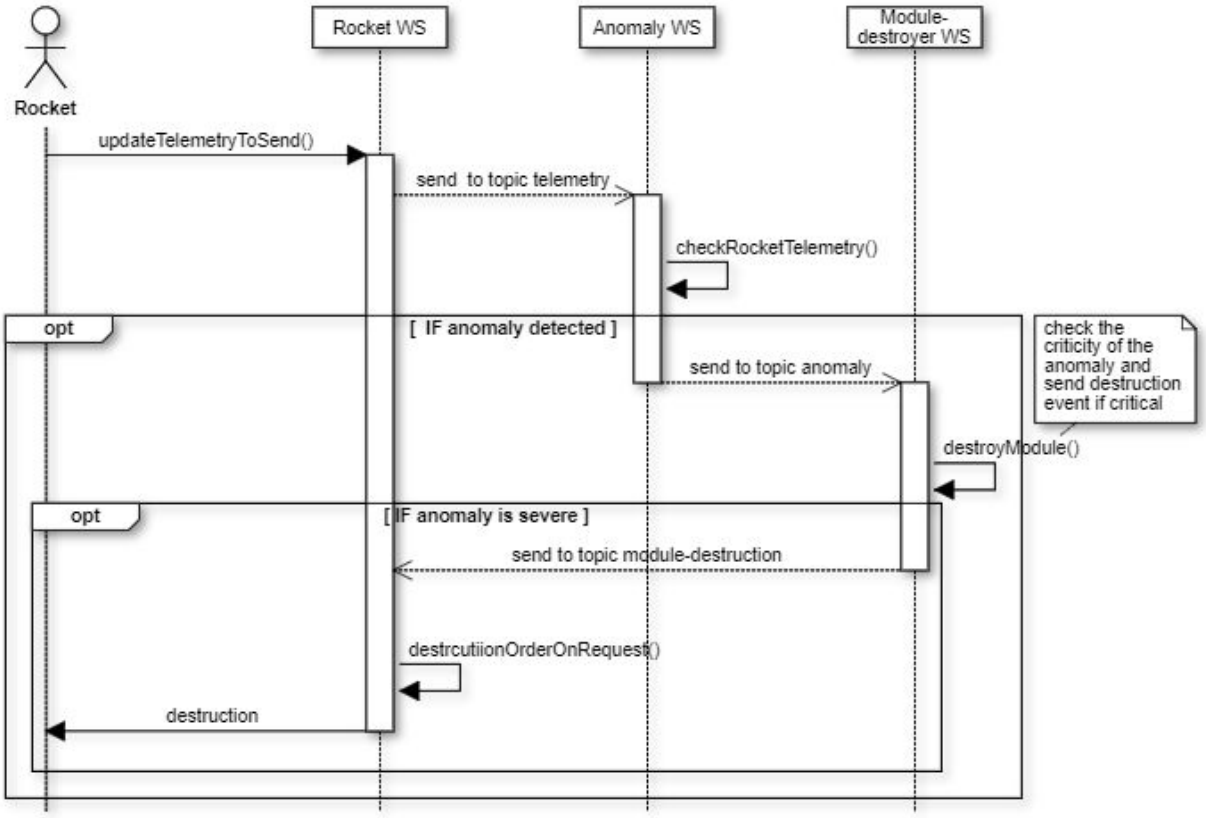

Figure 8 : diagramme de séquence de gestion des anomalies et destruction automatique

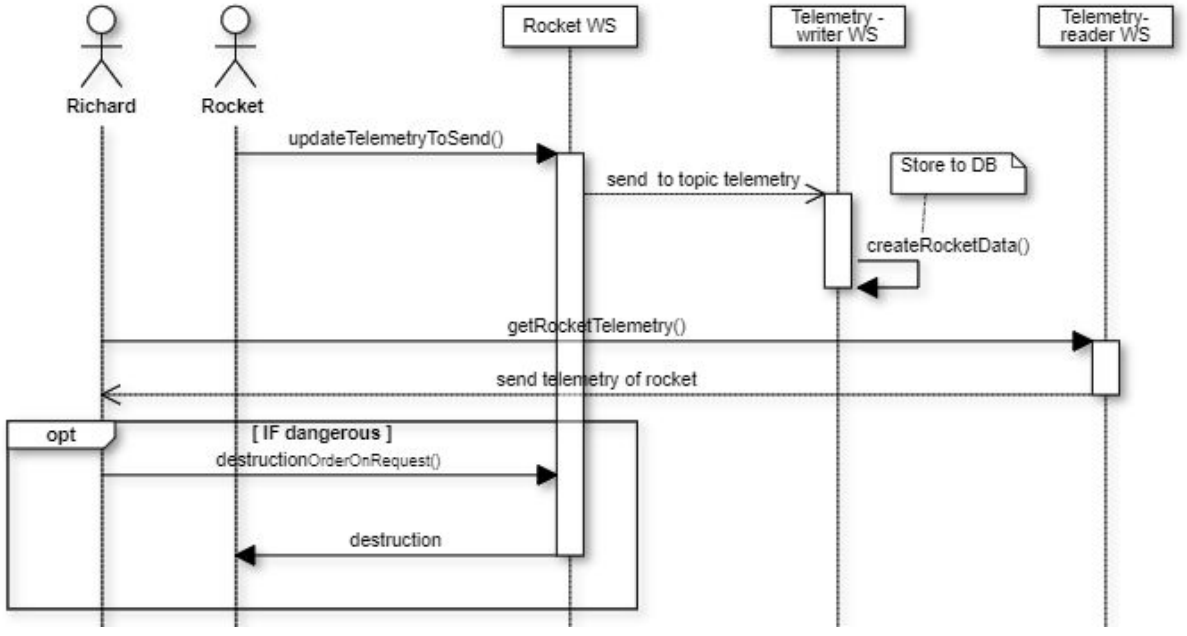

Figure 9 : diagramme de séquence de destruction de la fusée manuellement

Scénario 6 : MaxQ [User story: 12]

Lorsque la fusée est en plein vol, si le système s'aperçoit que la fusée est arrivée à une certaine altitude où on arrive dans la zone de MaxQ, alors le système donne un ordre à la fusée pour que

ce dernier réduit la puissance de ses propulseurs. Une fois cette zone passée, le système refait à nouveau un ordre pour augmenter la puissance des propulseurs.

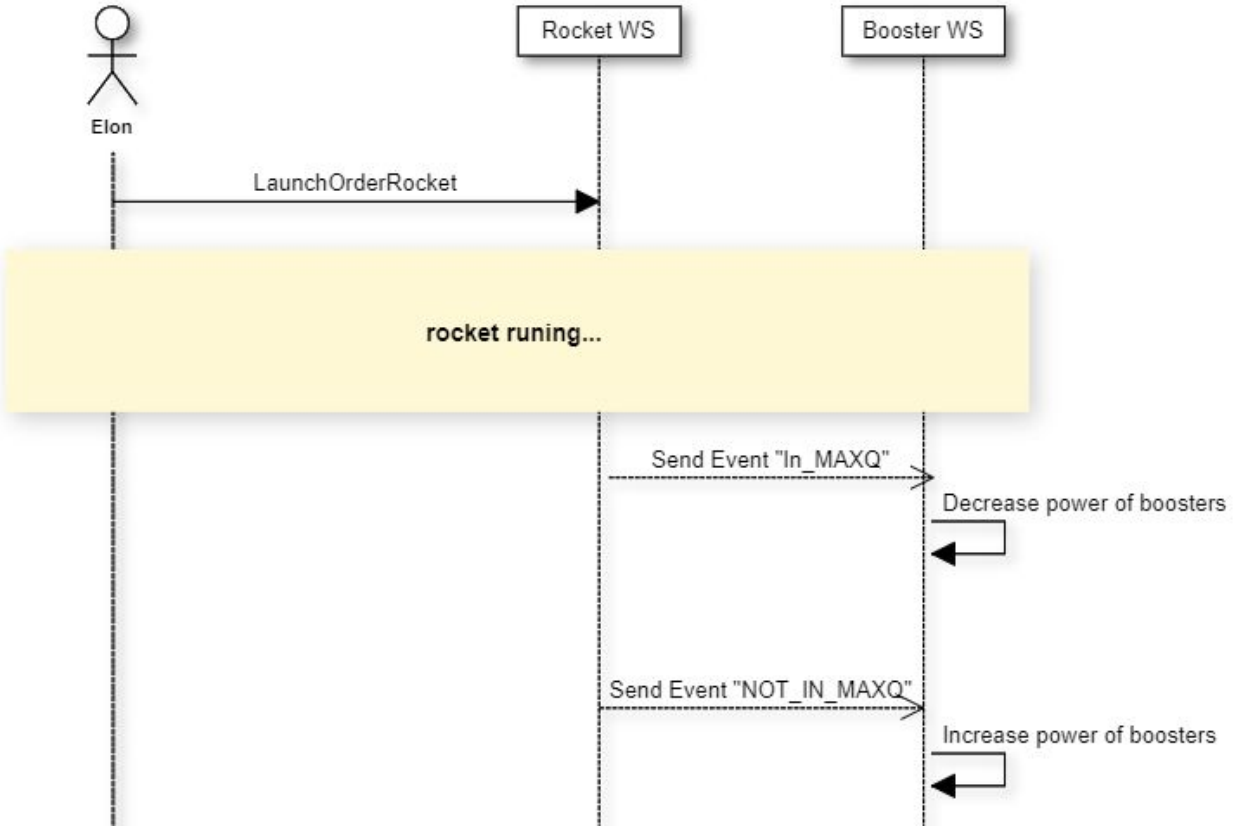

Figure 10 : diagramme de séquence de gestion de la phase MaxQ

Scénario 7 : Déploiement du payload [User story : 7, 11]

Lorsque la fusée arrive à la destination voulu, sur l'orbite de destination du payload, alors le payload est libéré. Cette action de libérer le payload entraîne, par la même occasion, la mise en fonctionnement de l'envoie de la télémétrie du payload vers la station au sol.

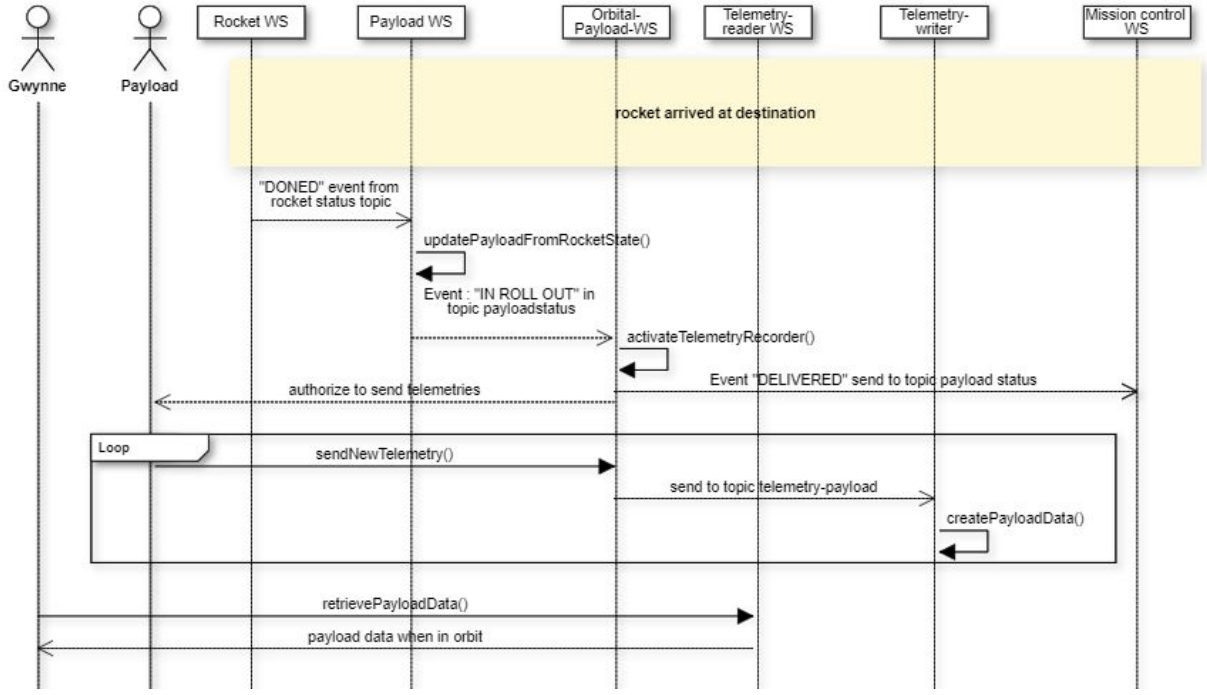

Figure 11 : diagramme de séquence de déploiement du payload

Scénario 8 : suivre la fusée [User story : 5, 13]

Après obtention des GO des départements Rocket et Weather, Richard peut autoriser le lancement de la fusée. Elon va enclencher le départ de la fusée. Au cours de toutes ces séquences la fusée va passer par plusieurs étapes, au cours desquelles les données de télémétrie sont envoyées au département Telemetry, ces données sont consultables à tout moment par Jeff.

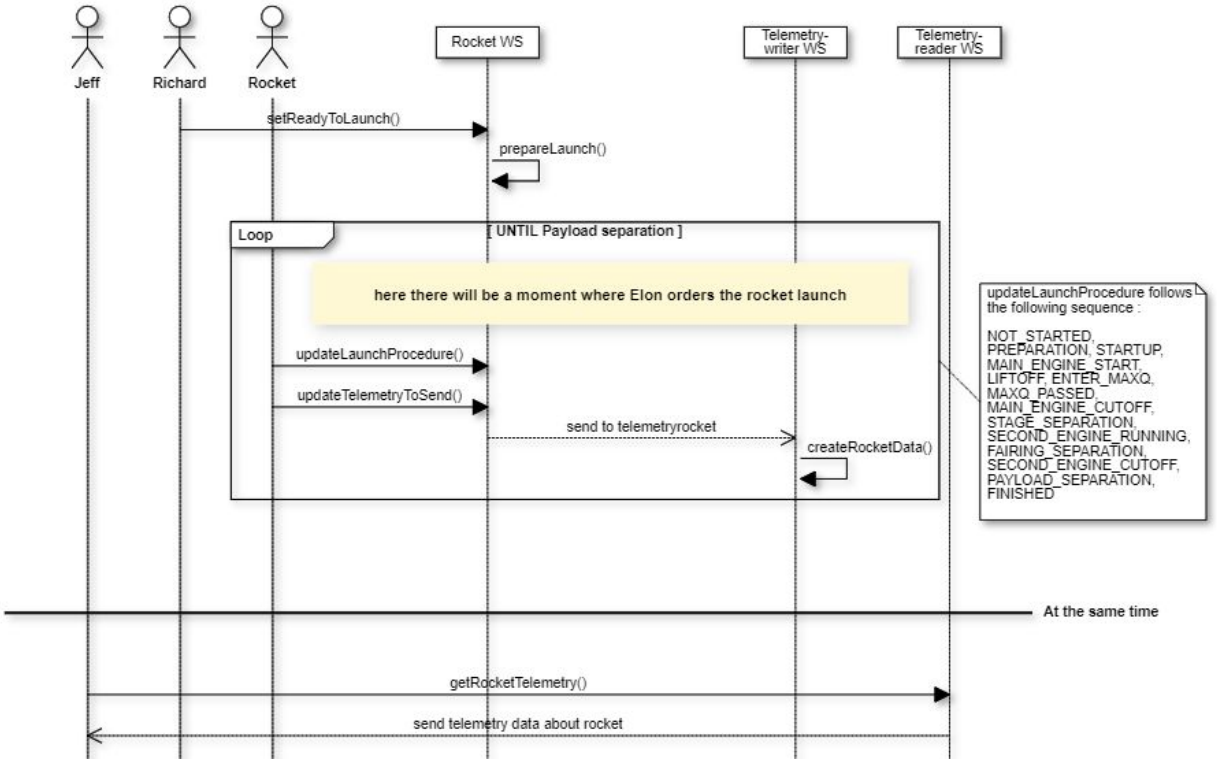

Figure 12 : diagramme de séquence de suivi de la fusée

Scénario 9: suivre la mission [User story: 14]

NB: par manque de temps et par soucis de rendre un projet stable, nous n'avons pas implémenté tous les producers possibles (ex: rocket launch event) pour pouvoir être consommé par le webservice MissionLogWriter. Malgré tout, nous avons l'architecture de base pour supporter ces scénarios.

Lorsqu'un événement important au cours de la mission se produit, un log spécifique à la mission est créé en fonction du type d'événement, puis Richard peut consulter ces logs qui contiennent des données relatives à la mission, ainsi que la date à laquelle il a été créé.

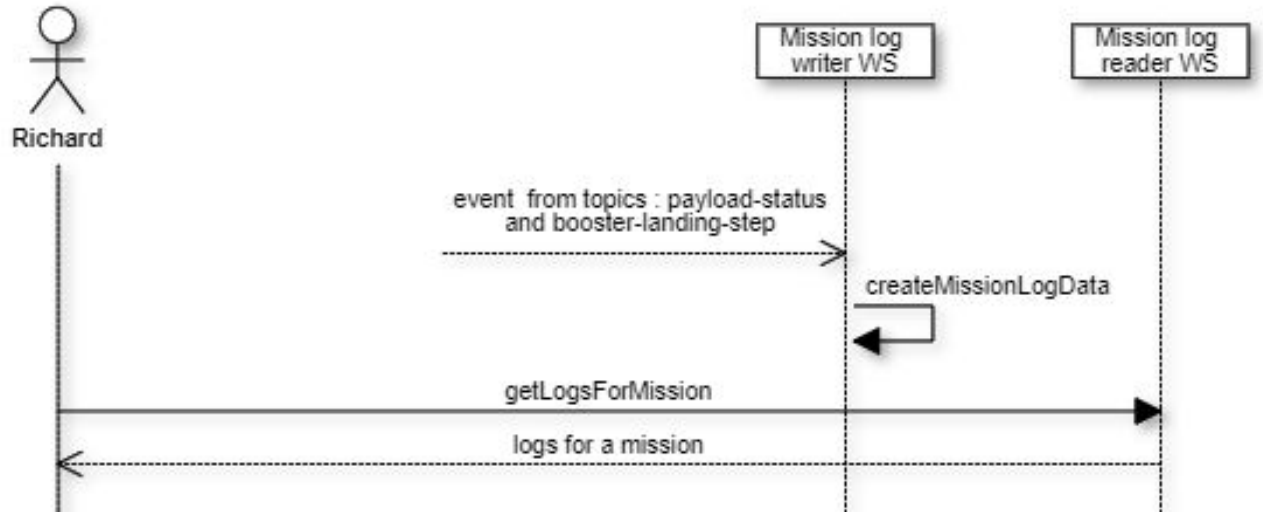

Figure 13 : diagramme de séquence de suivi de la mission

V. Ouverture et suite du projet

Rétrospectives

Nous avons pris des bonnes et des mauvaises décisions tout au long du projet. Les bonnes décisions étaient d'avoir une gestion de projet efficace (créer plusieurs branches, faire des tickets, etc…), d'être passé sur kafka très tôt et d'avoir appliquer les bonnes pratiques de développement. Les mauvaises décisions peuvent être trouvées sur le tableau ci-dessous avec les actions qui auraient dû être prises pour éviter d'avoir fait ce choix.

Mauvaises Décisions	**Actions**
Ne pas avoir testé plus tôt l'intégration des services.	Faire plus de tests fonctionnels, End-To-End.
Avoir pris du retard pour découper les services en micro-services.	Ne pas avoir peur du changement même si cela implique que des anciennes fonctionnalités développées "cassent".
Avoir fait des services trop gros (monolithe).	Faire une meilleure analyse du domaine pour mieux délimiter chacun des services isolés. Mieux mettre en place l'approche DDD. Séparation par domaine mais aussi par contexte.
Mauvaises séparations des US	Mieux faire les US pour impacter uniquement un micro-service à la fois. Si une US dois toucher plusieurs micro-services alors c'est une EPIC composée de plusieurs US.

Perspectives

Suite aux faiblesses et aux problèmes que l'on a trouvés, nous envisageons de séparer certains services qui ont encore beaucoup trop de responsabilités

MissionPreparation : Découpage en un nouveau service

- **MissionGoNoGo WebService :** Ce micro-service serait responsable d'envoyer le GO/NO-GO pour que la mission démarre

WebCasterLogWriter et Reader : Nous aurions aimé ajouter ces nouveaux services qui seraient similaires aux services de MissionLog. WebCasterLogWriter aurait la responsabilité de consommer des évènements Kafka différents en fonction de ce dont Marie veut être tenue au courant, et de générer un log qui plus lisible pour Marie, avec moins d'informations que dans MissionLog.

Booster : Découpage en trois nouveaux services :

- **BoosterHangar WebService :** Ce micro-service serait responsable de créer un nouveau booster
- **BoosterControl WebService :** Ce micro-service serait responsable de mettre à jour les boosters tout au long du vol
- **BoosterLanding WebService :** Ce micro-service serait responsable de mettre à jour la phase d'atterrissage de chaque booster

Rocket WebService : Découpage en deux nouveaux services

- **RocketHangar WebService :** Ce micro-service serait responsable de créer une nouvelle fusée
- **RocketControl WebService :** Ce micro-service serait responsable de mettre à jour la fusée tout au long du vol

Payload WebService : Découpage en trois nouveaux services :

- **PayloadHangar WebService :** Ce micro-service serait responsable de créer un nouveau payload
- **PayloadControl WebService :** Ce micro-service serait responsable de mettre à jour le payload tout au long du vol avant le déploiement
- **OrbitalTelemetryWriter WebService :** Ce micro-service serait responsable de d'écrire les télémétries du payload lorsque celui-ci est en déploiement

Voir sur ce lien une possible évolution de l'architecture

VI. Conclusions

Malgré les différents problèmes rencontrés et les difficultés, nous sommes assez satisfaits du travail fourni. Nous sommes beaucoup monté en compétence techniquement mais aussi d'un point de vue conception des architectures en micro-service puisque nous avons été confrontés aux différents problèmes qu'on pouvait rencontrer lors de l'établissement d'une telle architecture.